A Real-time Inversion Attack on the GMR-2 Cipher Used in the Satellite Phones

Jiao Hu, Ruilin Li (✉), and Chaojing Tang

School of Electronic Science and Engineering, National University of Defense Technology, Changsha, 410073, P. R. China
hujiao0614@163.com, securitylrl@gmail.com, cjtang@nudt.edu.cn

Abstract. The GMR-2 cipher is a kind of stream cipher currently being used in some Inmarsat satellite phones. It has been proven that such cipher can be cracked using only one frame known keystream but with a moderate executing times. In this paper, we present a new thorough security analysis of the GMR-2 cipher. We first study the inverse properties and the relationship of the cipher's components to reveal a bad one-way character of the cipher. Then by introducing a new concept called "valid key chain" according to the cipher's key schedule, we for the first time propose a real-time inversion attack using one frame keystream. This attack contains three phases: (1) table generation (2) dynamic table looks-up, filtration and combination (3) verification. Our analysis shows that, using the proposed attack, the exhaustive search space for the 64-bit encryption key can be reduced to about 2^{13} when one frame (15 bytes) keystream is available. Compared with previous known attacks, this inversion attack is much more efficient. Finally, the proposed attack are carried out on a 3.3GHz platform, and the experimental results demonstrate that the 64-bit encryption-key could be recovered in around 0.02s on average.

Keywords: Satellite Phone, Stream Cipher, GMR-2, Cryptanalysis, Inversion Attack

1 Introduction

1.1 Backgrounds and the GMR-2 Cipher

With the rapid evolution and development of 4G technologies, mobile phone systems are available worldwide nowadays, still it is difficult to build a complete mobile network in some remote areas, such as outlying desert areas, oceans, and mountains. Thus, to fill the gaps left behind by radio-based technologies, satellite phones have been widely used in these areas. Currently, the commonly used satellite communication standards are mainly developed by international standards organization ETSI [7], including the GMR-1 standard and the GMR-2 standard. For instance, Thuraya phones are based on the GMR-1 standard, while the Inmarsat phones adopt GMR-2 standard.

Given that the confidentiality is a very crucial aspect in satellite communications, the encryption algorithms in the satellite phones should be strong enough to withstand various eavesdropping risks. For mobile application scenario, many symmetric ciphers were developed and adopted as the cryptographic components for secure communications, *e.g.*, A5, SNOW, and ZUC, and their security were sufficiently evaluated in past years [2, 6, 11, 12, 14–16]. However, the GMR cryptographic algorithms are not included in the officially published GMR standards, and the details of these satellite cipher algorithms were non-public until the German research team Driessen *et al.* uncovered the GMR-1 and the GMR-2 cipher by reverse engineering in 2012 [4, 5]. Their analysis results illustrate that both two ciphers are stream ciphers. In particular, the GMR-1 cipher is a proprietary variant of GSM A5/2 algorithm [4], thus the cryptanalytic methods against the A5/2 algorithm [1, 3] can almost be well-adopted to it. The GMR-2 cipher is an entirely newly designed stream cipher, however, it has been found to be insecure for two types of known plaintext attacks. Driessen *et al.* proposed a known plaintext attack against it for the first time based on the read-collision technique [4] according to the key-scheduling features of the GMR-2 cipher. The time complexity of such attack is about 2^{18} with approximately $50 \sim 65$ bytes of keystream. Li *et al.* further put forward a low data complexity attack method called the dynamic guess-and-determine attack [13] which can break the GMR-2 cipher by guessing about 28 bits on average when 15 bytes of keystream are available.

1.2 Main Contribution and the Outline

Generally speaking, stream ciphers firstly generate keystreams by implementing a series of complex cryptographic transformation on the initial vectors and the encryption-key, and then XOR the keystreams with plaintexts to obtain the ciphertexts. Therefore, to resist known plaintext attack, a vital requirement of stream ciphers is the one-way property, *i.e.*, it must be difficult for the adversary to derive the encryption-key from the keystream through inversion procedure. According to [8–10], Golic *et al.* proposed an inversion attack methods against the keystream generator consisting of a linear feedback shift register and a nonlinear filter, and proved the effectiveness of such attack in some cases.

In this paper, we study the *inverse properties* of the GMR-2 cipher to show a bad one-way character of such cipher, then by introducing a new concept "valid key chain", we propose what we call *the inversion attack* against the GMR-2 cipher. Our proposed attack consists of three major phases: (1) table generation, (2) dynamic table looks-up, filtration and combination, (3) verification. With the help of an extra 6KB memory storage, this attack can reduce the exhaustive search space from 2^{64} to about 2^{13} on average when one frame (15 bytes) keystream is available. This indicates that the inversion attack is very efficient and practical which could lead to a real time crack on the GMR-2 cipher. The experimental results on a 3.3GHz platform demonstrate that the 64-bit encryption-key can be completely retrieved in around 0.02s.

This paper is organized as follows: a brief introduction to the GMR-2 cipher is recalled in Section 2. Section 3 and 4 analyse the inverse properties as well as the relationship of the the three components of the GMR-2 cipher. Section 5 proposes the attack strategy and details the attack procedure. The experimental results and the attack complexity will be subsequently analysed in Section 6. Finally, Section 7 gives a concise summary of this paper.

2 Description of the GMR-2 Cipher

The GMR-2 cipher is a kind of stream cipher with 64-bit key-length. As shown in Fig. 1, the internal states of the cipher include a 8-byte shift register $S = (S_7, S_6, \cdots, S_0)$, a 8-byte encryption-key register $K = (K_7, K_6, \cdots, K_0)$, a counter $c \in \{0, 1, \cdots, 7\}$, and a toggle-bit $t \in \{0, 1\}$. They are transformed through three components $\mathcal{F}$, $\mathcal{G}$, and $\mathcal{H}$. At each clock l, the cipher generates one byte keystream, which we denote by Z_l.

The $\mathcal{F}$-component can be treated as a key schedule part of the GMR-2 cipher, it combines two bytes of the encryption-key with the previous output (a key-stream byte) to compute a 12-bit output. The $\mathcal{G}$-component is designed for mixing purpose, it is a linear function with 12-bit input and 12-bit output. The $\mathcal{H}$-component is a nonlinear filter, it consists of two parallel DES S-boxes with 16-bit input and 8-bit output. The following subsections describe these three components in detail.

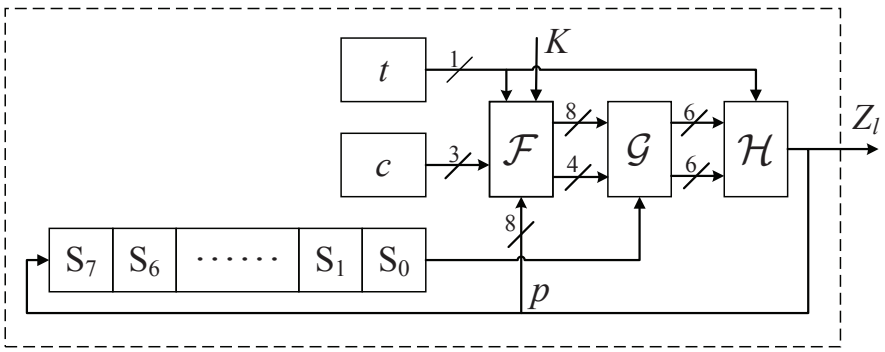


Fig. 1. Overall Structure of the GMR-2 Cipher

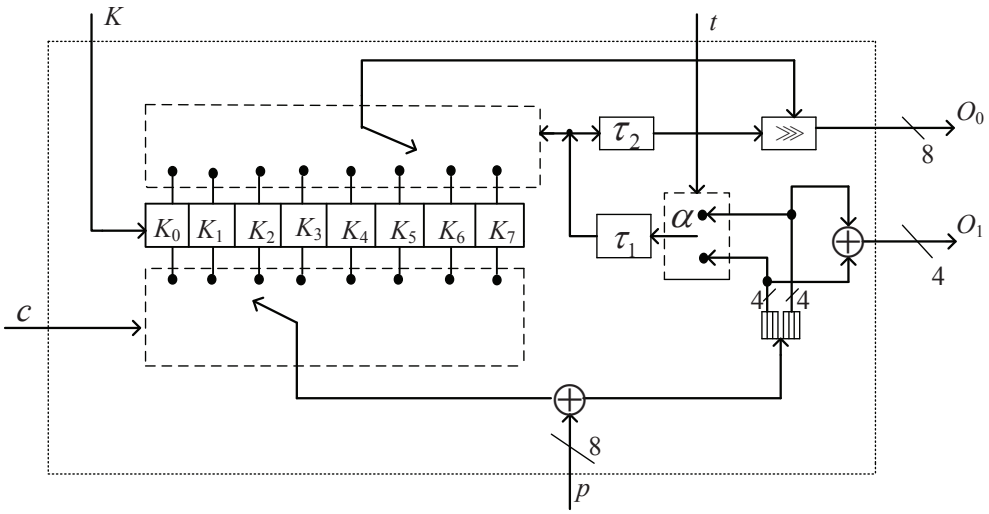


Fig. 2. The structure of $\mathcal{F}$-component

Table 1. Definition of τ_1 and τ_2

α	$\tau_1(\alpha)$	$\tau_2(\tau_1(\alpha))$	α	$\tau_1(\alpha)$	$\tau_2(\tau_1(\alpha))$
(0,0,0,0)	2	6	(1,0,0,0)	3	7
(0,0,0,1)	5	3	(1,0,0,1)	**0**	**4**
(0,0,1,0)	**0**	**4**	(1,0,1,0)	6	2
(0,0,1,1)	6	2	(1,0,1,1)	1	5
(0,1,0,0)	3	7	(1,1,0,0)	5	3
(0,1,0,1)	7	1	(1,1,0,1)	7	1
(0,1,1,0)	**4**	**4**	(1,1,1,0)	**4**	**4**
(0,1,1,1)	1	5	(1,1,1,1)	2	6

2.1 Components of the GMR-2 cipher

***$\mathcal{F}$*-component** As the most interesting part of the cipher, the internal structure of $\mathcal{F}$-component is depicted in Fig. 2. The 8-byte encryption-key $K = (K_7, K_6, \ldots, , K_0)$ is fed into a 64-bit resister and it is unchanged during the whole execution of the cipher. At each clock, the $\mathcal{F}$-component selects two key bytes (one from the lower side and the other from the upper side) for further computation, and the procedure can be described formally as follows:

Assume the cipher is executed at the l-th clock, besides the 8-byte encryption-key K, the inputs of the $\mathcal{F}$-component also contain three variables c, t, and p, where $c = l \mod 8$ is a counter ranging from 0 to 7 sequentially and repeatedly, $t = c \mod 2$ is a toggle bit, and $p = (p_7, p_6, \ldots, p_0) \in \{0,1\}^8$ is a feedback keystream byte that has already been generated in the last clock. We simply use $p = Z_{l-1}$ to denote the keystream byte that was generated at the previous (the $(l-1)$-th) clock. The outputs of the $\mathcal{F}$-component consist of an 8-bit O_0 and a 4-bit O_1 with the following definitions:

$$\begin{cases} O_0 = (K_{\tau_1(\alpha)} \ggg \tau_2(\tau_1(\alpha)))_8 \\ O_1 = ((((K_c \oplus p) \gg 4) \& \mathsf{0x0F}) \oplus ((K_c \oplus p) \& \mathsf{0x0F}))_4 \end{cases} \tag{1}$$

where α is defined by

$$\alpha = \mathcal{N}(t, K_c \oplus p) = \begin{cases} ((K_c \oplus p) \& \mathsf{0x0F}))_4 & \text{if} \quad t = 0 \\ (((K_c \oplus p) \gg 4) \& \mathsf{0x0F})_4 & \text{if} \quad t = 1 \end{cases} \tag{2}$$

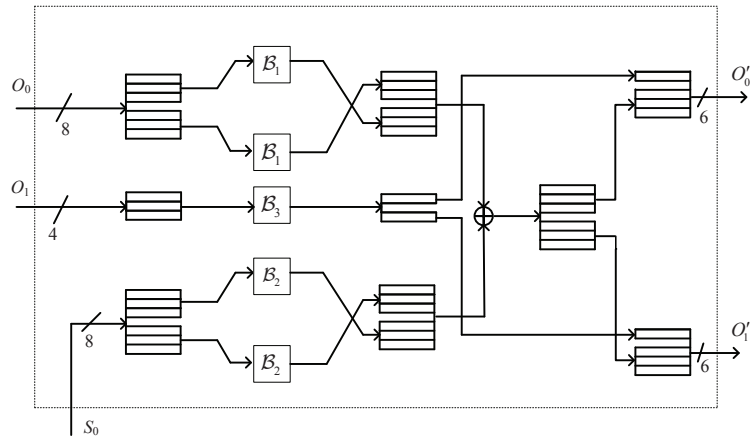


Fig. 3. The structure of $\mathcal{G}$-component

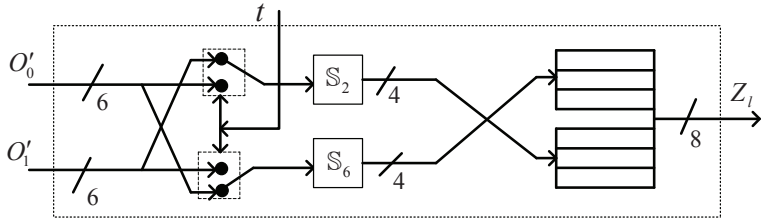


Fig. 4. The structure of $\mathcal{H}$-component

and $\tau_1 : \{0,1\}^4 \to \{0,1\}^3$, $\tau_2 : \{0,1\}^3 \to \{0,1\}^3$ are two functions implemented via table-looksup as shown in Table 1.

$\mathcal{G}$-component Fig. 3 illustrates the structure of the $\mathcal{G}$-component, where $\mathcal{B}_1$, $\mathcal{B}_2$, and $\mathcal{B}_3$ are all linear functions. $\mathcal{G}$-component gets the outputs of the $\mathcal{F}$-component (O_0 and O_1) and one-byte state register $S_0 = (S_{0,7}, S_{0,6}, \ldots, S_{0,0})$ as its inputs, and after a series of linear transformation, transposition, and XOR operations, it outputs two 6-bit outputs O'_0 and O'_1, which can be expressed as follows:

$$\begin{cases} O'_0 = (O_{0,7} \oplus O_{0,4} \oplus S_{0,5}, O_{0,7} \oplus O_{0,6} \oplus O_{0,4} \oplus S_{0,7}, O_{0,7} \oplus S_{0,4}, \\ \qquad O_{0,5} \oplus S_{0,6}, O_{1,3} \oplus O_{1,1} \oplus O_{1,0}, O_{1,3} \oplus O_{1,0})_6 \\ O'_1 = (O_{0,3} \oplus O_{0,0} \oplus S_{0,1}, O_{0,3} \oplus O_{0,2} \oplus O_{0,0} \oplus S_{0,3}, O_{0,3} \oplus S_{0,0}, \\ \qquad O_{0,1} \oplus S_{0,2}, O_{1,2}, O_{1,0})_6 \end{cases} \tag{3}$$

$\mathcal{H}$-component $\mathcal{H}$-component gets the two 6-bit outputs of the $\mathcal{G}$-component as its input. Fig. 4 shows the structure of $\mathcal{H}$-component which is composed of two 6-in and 4-out S-boxes ($\mathbb{S}_2$ and $\mathbb{S}_6$) used in DES algorithm. However, these two S-boxes have been reordered to account for the different addressing. Assume that the 6-bit input of the S-box is $(x_5, x_4, x_3, x_2, x_1, x_0)_2$, for GMR-2 cipher, the most-significant 4-bits (x_5, x_4, x_3, x_2) determine the column index of the S-box while the least-significant 2-bits (x_1, x_0) select the S-box row index. Finally, depending on the toggle-bit t, the output one-byte keystream can be defined by:

$$Z_l = \begin{cases} (\mathbb{S}_2(O'_1), \mathbb{S}_6(O'_0))_8 & \text{if} \quad t = 0 \\ (\mathbb{S}_2(O'_0), \mathbb{S}_6(O'_1))_8 & \text{if} \quad t = 1 \end{cases} \tag{4}$$

2.2 Mode of operation

As mentioned in [4], we can now describe the mode of operation for the GMR-2 cipher. When the cipher is clocked at the l-th time, the state of the GMR-2 cipher will be changed as follows:

- The cipher generates one-byte keystream Z_l based on the current value of the state-register S, the counter c and the toggle bit $t = c \mod 2$.
- The counter c is incremented by one, and when 8 is reached for c, c is reset to 0.
- The state-register S is shifted by one byte to the right, thus S_i=S_{i+1}, for $i = 0, 1, 2, \ldots, 6$, and S_7=Z_l. Meanwhile, $p = S_7 = Z_l$ is passed to the $\mathcal{F}$-component as the input parameter for the next iteration (the $(l+1)$-th clock).

The GMR-2 cipher is operated in two modes: the initialization mode and the generation mode.

Initialization Mode. In the initialization phase, the following steps are performed:

- The counter $c = 0$ and the toggle-bit $t = 0$.
- The 64-bit encryption-key is written into the resister in the $\mathcal{F}$-component.
- The state-register S is initialized with a 22-bit frame-number N according to a special rule, which is not detailed here as it is irrelevant with our attack.
- After c, t, S have been initialized, the cipher is clocked 8 times, but the resulting keystream is *discarded*.

Generation Mode. After the initialization is finished, the cipher is switched into generation mode to produce and output actual keystream bytes. We use $Z_l^{(N)}$ to denote the l-th keystream byte after initialization with the frame number N. For each frame number N, the cipher will operate 15 clocks and generate a 15-byte keystream. After that, the frame number N automatically increases by 1 and the state-register is re-initialized with the new frame number, and then the cipher will generate another 15-byte keystream. Assuming the frame number starts from 0, the actual keystream Z' is made up of blocks of 15 bytes that are concatenated as follows:

$$Z' = \left(Z_0^{(0)}, Z_1^{(0)}, \ldots, Z_{14}^{(0)}; Z_0^{(1)}, Z_1^{(1)} \cdots Z_{14}^{(1)}; Z_0^{(2)}, \ldots\right) \tag{5}$$

3 Inverse Properties of the GMR-2 Cipher's Components

The GMR-2 cipher consists of three components, in which the $\mathcal{F}$-component plays a role of key schedule, the $\mathcal{G}$-component acts as a linear transformation, and the $\mathcal{H}$-component implements a nonlinear transformation. Both the cryptanalytic methods proposed in [4, 13] originate from the forward analysis of the GMR-2 cipher, whereas our proposed inversion attack is inspired from the backward analysis, *i.e.*, we try to reverse the encryption procedure to deduce the encryption-key from the output keystream directly. Thus in this section, we will first study the inverse properties of the three components which are related to our later analysis.

3.1 Inverse Property of the $\mathcal{H}$-component

$\mathcal{H}$-component is parallelly composed of two S-boxes, and it selects the column and row indices of the two S-boxes through the toggle-bit t, as show in Eq. 4. In fact, We can extract the relationship between the input (O'_0, O'_1) of $\mathcal{H}$ and the output Z_l of $\mathcal{H}$ (the keystream byte) by "inverting" the two S-boxes. Thus, we have the following proposition:

Proposition 1 *For $\mathcal{H}$-component, if the row index and the output of an S-box are known, the column index can be uniquely determined; If only the outputs of both S-boxes are known, there will be $4 \times 4 = 16$ different corresponding row/column indices.*

3.2 Inverse Property of the $\mathcal{G}$-component

Assume the shift register S_0 is known, then $\mathcal{G}$-component can be represented by an affine transformation. We focus on how to extract the inputs O_0 and O_1 of $\mathcal{G}$, given the output O'_0 and O'_1 along with S_0. According to [13] and Eq. 3, the link[1] between the input and output of the $\mathcal{G}$-component can be expressed by

$$\begin{cases} \boldsymbol{y}_1 = \boldsymbol{W}_1 \cdot \boldsymbol{x}_1 \oplus \boldsymbol{Q}' \cdot \boldsymbol{v} \\ \boldsymbol{y}_2 = \boldsymbol{W}_2 \cdot \boldsymbol{x}_2 \end{cases} \tag{6}$$

where

$$\boldsymbol{W}_1 = \begin{pmatrix} \boldsymbol{A} & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{A} \end{pmatrix}, \boldsymbol{W}_2 = (\boldsymbol{B}), \boldsymbol{Q}' = \begin{pmatrix} \boldsymbol{C} & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{C} \end{pmatrix},$$

$$\boldsymbol{A}= \begin{pmatrix} 1\,0\,0\,1 \\ 1\,1\,0\,1 \\ 1\,0\,0\,0 \\ 0\,0\,1\,0 \end{pmatrix} \boldsymbol{B}= \begin{pmatrix} 1\,0\,1\,1 \\ 1\,0\,0\,1 \\ 0\,1\,0\,0 \\ 0\,0\,0\,1 \end{pmatrix}, \boldsymbol{C} = \begin{pmatrix} 0\,0\,1\,0 \\ 1\,0\,0\,0 \\ 0\,0\,0\,1 \\ 0\,1\,0\,0 \end{pmatrix}, \boldsymbol{0}= \begin{pmatrix} 0\,0\,0\,0 \\ 0\,0\,0\,0 \\ 0\,0\,0\,0 \\ 0\,0\,0\,0 \end{pmatrix}$$

and

$$\begin{cases} \boldsymbol{y}_1 = (O'_{0,5}, O'_{0,4}, O'_{0,3}, O'_{0,2}, O'_{1,5}, O'_{1,4}, O'_{1,3}, O'_{1,2})^T \\ \boldsymbol{y}_2 = (O'_{0,1}, O'_{0,0}, O'_{1,1}, O'_{1,0})^T \end{cases},$$

$$\begin{cases} \boldsymbol{x}_1 = (O_{0,7}, O_{0,6}, O_{0,5}, O_{0,4}, O_{0,3}, O_{0,2}, O_{0,1}, O_{0,0})^T \\ \boldsymbol{x}_2 = (O_{1,3}, O_{1,2}, O_{1,1}, O_{1,0})^T \\ \boldsymbol{v} = (S_{0,7}, S_{0,6}, S_{0,5}, S_{0,4}, S_{0,3}, S_{0,2}, S_{0,1}, S_{0,0})^T \end{cases}.$$

In the above formulas, $\boldsymbol{x}_1$, $\boldsymbol{x}_2$ and $\boldsymbol{v}$ are used to represent O_0, O_1 and S_0, which is the input of $\mathcal{G}$, and $(\boldsymbol{y}_1, \boldsymbol{y}_2)$ is used to represent a simple permutation of (O'_0, O'_1) which is the output of $\mathcal{G}$. By carefully observing the $\mathcal{H}$-component, we can see that $\boldsymbol{y}_1$ corresponds to column indices of the two S-boxes, and $\boldsymbol{y}_2$ corresponds to row indices of the two S-boxes.

Now if we treat $\boldsymbol{y}_1$, $\boldsymbol{y}_2$ and $\boldsymbol{v}$ (thus O'_0, O'_1, and S_0) as known values, while $\boldsymbol{x}_1$ and $\boldsymbol{x}_2$ (thus O_0 and O_1) as unknown variables, the first/second formula in Eq. 6 can be regarded as a system of linear equations with 8/4 variables. Since both $\boldsymbol{A}$ and $\boldsymbol{B}$ are invertible matrices, we get

$$\begin{cases} \boldsymbol{x}_1 = \boldsymbol{W}_1^{-1} \cdot \boldsymbol{y}_1 \oplus \boldsymbol{Q} \cdot \boldsymbol{v} \\ \boldsymbol{x}_2 = \boldsymbol{W}_2^{-1} \cdot \boldsymbol{y}_2 \end{cases} \tag{7}$$

where

$$\boldsymbol{W}_1^{-1}= \begin{pmatrix} \boldsymbol{A}^{-1} & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{A}^{-1} \end{pmatrix}, \boldsymbol{W}_2^{-1} = (\boldsymbol{B}^{-1}), \boldsymbol{Q} = \boldsymbol{W}_1^{-1} \cdot \boldsymbol{Q}' = \begin{pmatrix} \boldsymbol{A}^{-1} \cdot \boldsymbol{C} & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{A}^{-1} \cdot \boldsymbol{C} \end{pmatrix},$$

$$\boldsymbol{A}^{-1} = \begin{pmatrix} 0\,0\,1\,0 \\ 1\,1\,0\,0 \\ 0\,0\,0\,1 \\ 1\,0\,1\,0 \end{pmatrix}, \boldsymbol{B}^{-1} = \begin{pmatrix} 0\,1\,0\,1 \\ 0\,0\,1\,0 \\ 1\,1\,0\,0 \\ 0\,0\,0\,1 \end{pmatrix}, \boldsymbol{A}^{-1} \cdot \boldsymbol{C} = \begin{pmatrix} 0\,0\,0\,1 \\ 1\,0\,1\,0 \\ 0\,1\,0\,0 \\ 0\,0\,1\,1 \end{pmatrix}.$$

Therefore, we have the following proposition:

Proposition 2 *For $\mathcal{G}$-component, if O'_0, O'_1 and S_0 (thus $\boldsymbol{y}_1$, $\boldsymbol{y}_2$ and $\boldsymbol{v}$) are known values, then O_0 and O_1 (thus $\boldsymbol{x}_1$ and $\boldsymbol{x}_2$) can be calculated directly from Eq. 7. Specifically, O_0 (thus $\boldsymbol{x}_1$) is uniquely determined by $\boldsymbol{y}_1$, and O_1 (thus $\boldsymbol{x}_2$) is uniquely determined by $\boldsymbol{y}_2$.*

[1] Note that the definition of the variable $\boldsymbol{v}$ in Eq. 6 in this paper is different from the one in Ref [13]. In fact, $\boldsymbol{Q}' \cdot \boldsymbol{v}$ in this paper is equivalent to $\boldsymbol{v}_1$ as defined in Ref [13].

3.3 Inverse Property of the $\mathcal{F}$-component

$\mathcal{F}$ is the only component that relates to the original encryption-key bytes, thus it is critical for us to analyze. At each clock, $\mathcal{F}$-component selects K_c and $K_{\tau_1(\alpha)}$ for further computation. K_c is simply selected by the counter c, while $K_{\tau_1(\alpha)}$ is selected by the subscript $\tau_1(\alpha)$ which can be determined according to Eq. 2 and Table 1.

The inverse analysis of the $\mathcal{F}$-component aims at deducing the above two selected key bytes from the known output (O_0, O_1) and the feedback byte p. Rewriting the second formula of Eq. 1, K_c can be expressed by O_1 and p as follows

$$\begin{cases} K_{c,7} \oplus K_{c,3} = O_{1,3} \oplus p_7 \oplus p_3 \\ K_{c,6} \oplus K_{c,2} = O_{1,2} \oplus p_6 \oplus p_2 \\ K_{c,5} \oplus K_{c,1} = O_{1,1} \oplus p_5 \oplus p_1 \\ K_{c,4} \oplus K_{c,0} = O_{1,0} \oplus p_4 \oplus p_0 \end{cases} \tag{8}$$

For simplicity, let's denote

$$\begin{cases} \boldsymbol{k}_h = (K_{c,7}, K_{c,6}, K_{c,5}, K_{c,4})^T \\ \boldsymbol{k}_l = (K_{c,3}, K_{c,2}, K_{c,1}, K_{c,0})^T \\ \boldsymbol{p}_h = (p_7, p_6, p_5, p_4)^T \\ \boldsymbol{p}_l = (p_3, p_2, p_1, p_0)^T \end{cases}$$

then Eq. 8 becomes

$$\boldsymbol{k}_h \oplus \boldsymbol{k}_l = O_1 \oplus \boldsymbol{p}_h \oplus \boldsymbol{p}_l. \tag{9}$$

Therefore, for $i = 0, 1, 2, 3$, when $O_{1,i} \oplus p_{i+4} \oplus p_i = 0$, the candidate for $(K_{c,i+4}, K_{c,i})$ is selected from $\{(0,0), (1,1)\}$, and when $O_{1,i} \oplus p_{i+4} \oplus p_i = 1$, the candidate can be only selected from $\{(0,1), (1,0)\}$. This implies that given O_1 and p, Eq. 8 has 16 solutions for K_c.

Similarly, rewriting the first formula of Eq. 1, $K_{\tau_1(\alpha)}$ can be obtained from O_0 by

$$K_{\tau_1(\alpha)} = O_0 \lll \tau_2(\tau_1(\alpha)), \tag{10}$$

where α is related to K_c and p, and can be calculated on the basis of the Eq. 2. That is to say, we can get the value of α through p and the most/least-significant 4-bits of K_c. This leads to the following proposition:

Proposition 3 *For $\mathcal{F}$-component, if O_1 and p are known, then all possible values of K_c can be narrowed down from 2^8 to 2^4 according to Eq. 8; If O_0, p, and K_c are known, the input key byte $K_{\tau_1(\alpha)}$ can be uniquely retrieved by Eq. 10.*

Now we have obtained three inverse properties of GMR-2 cipher's components as described in Proposition $1, 2, 3$. At the end of this section, let's briefly discuss the links among these inverse components as depicted in Fig. 5. Given the start point - a keystream byte $Z_l^{(N)}$ at the l-th clock with frame number N, and assume the feedback byte S_0 and p is known. Then through $\mathcal{H}^{-1}$-component, 16 possible values of (O_0', O_1') will be obtained. This is followed by passing through $\mathcal{G}^{-1}$-component, which results in 16 different values of (O_0, O_1). And finally after $\mathcal{F}^{-1}$-component, each (O_0, O_1) will deduce 16 candidates for $(K_c, K_{\tau_1(\alpha)})$. In total, one can obtain at most $16 \times 16 = 256$ possible values for $(K_c, K_{\tau_1(\alpha)})$. The detail analysis will be described in the following section.

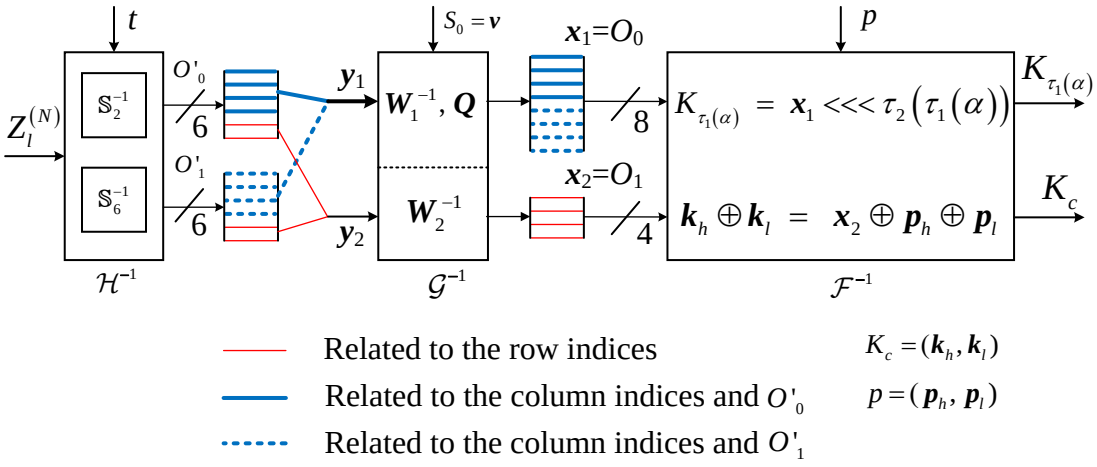


Fig. 5. Links among the Three Inverse Components

4 Inverse Properties of the GMR-2 Cipher

In this section, we will analyze how these three inverse components interact with each other and show the links between the keystream bytes and the original encryption key bytes.

Given a frame number N, let $S_i^{(l)}$ denote the state of S_i at the l-th clock and $Z_l^{(N)}$ denote the keystream byte at the l-th clock with N-th frame in the keystream generation phrase, then for $8 \le l \le 14$ we have

$$S_0^{(l)} = Z_{l-8}^{(N)} \quad \text{and} \quad p = S_7^{(l)} = Z_{l-1}^{(N)},$$

which demonstrates that $S_0^{(l)}$ is equal to the keystream byte generated 8 clocks before, and p is equal to the last keystream byte. Hence, for $8 \le l \le 14$, both $S_0^{(l)}$ and p are known to us, so is the vector $\boldsymbol{v}$ as previously defined. To this end, we only focus on the cipher at the $(c+8)$-th clock with $0 \le c \le 6$ in the following analysis. The main results are the following two theorems.

Theorem 1. *At the $(c+8)$-th clock with $0 \le c \le 6$, if K_c is known, then the corresponding encryption-key byte $K_{\tau_1(\alpha)}$ can be uniquely determined by the current keystream byte $Z_{c+8}^{(N)}$.*

Proof. Since p is known at the $(c+8)$-th clock, from Eq. 2, knowing K_c can help us to calculate α, as well as $\tau_1(\alpha)$ and $\tau_2(\tau_1(\alpha))$ via looking up Table 1. Meanwhile, O_1 (thus $\boldsymbol{x}_2$) can be obtained from Eq. 1, based on which $\boldsymbol{y}_2$ can be calculated from Eq. 6. Due to Proposition 1, $\boldsymbol{y}_1$ which corresponds to the column indices for the two S-boxes can be uniquely determined from $Z_{c+8}^{(N)}$ and the row indices $\boldsymbol{y}_2$. Consequently, the value of O_0 can be uniquely determined by Proposition 2. At last, with the help of Proposition 3, $K_{\tau_1(\alpha)}$ can be calculated definitely from O_0, K_c and p. □

Theorem 2. *At the $(c+8)$-th clock with $0 \le c \le 6$, each keystream byte $Z_{c+8}^{(N)}$ exactly corresponds to 256 possible values of the triple $(K_c, K_{\tau_1(\alpha)}, \tau_1(\alpha))$, where K_c is ranged from 0 to 255.*

Proof. First, according to Proposition 1 and Proposition 2, each keystream byte $Z_{c+8}^{(N)}$ corresponds to 16 different O_1, and for each O_1, Proposition 3 further indicates the existence of 16 different candidates for K_c.

Next, by contradiction, we can prove that the candidates for K_c obtained by different O_1 will be different from each other. That is to say, assuming that $O_1^{(i)} \neq O_1^{(j)}$ holds, one can declaim that the candidates $K_c^{(i)}$ and $K_c^{(j)}$ that are derived from $O_1^{(i)}$ and $O_1^{(j)}$ must be different. Otherwise if $K_c^{(i)} = K_c^{(j)}$, then Eq. 1 indicates $O_1^{(i)} = O_1^{(j)}$, which contradicts the hypothesis.

The above two steps demonstrate that each keystream byte $Z_{c+8}^{(N)}$ exactly corresponds to 256 values of K_c. Moreover, through Theorem 1, $\tau_1(\alpha)$ and $K_{\tau_1(\alpha)}$ can be further uniquely obtained from the K_c and $Z_{c+8}^{(N)}$, which completes this proof. □

Theorem 2 tells us that each $Z_{c+8}^{(N)}$ with $0 \leq c \leq 7$ can deduce several triples $(K_c, K_{\tau_1(\alpha)}, \tau_1(\alpha))$. Next, let's discuss how these triples can be further used to get new information on K. Here we list two rules that are very crucial for our attack.

Rule 1 Given one triple $(K_c, K_{\tau_1}(\alpha), \tau_1(\alpha))$ corresponding to the keystream byte $Z_{c+8}^{(N)}$ with $0 \leq c \leq 6$, if $\tau_1(\alpha) = c$, we can compare $K_{\tau_1(\alpha)}$ with K_c:

- If $K_c = K_{\tau_1(\alpha)}$, it indicates that such K_c can be regarded as a candidate;
- If $K_c \neq K_{\tau_1(\alpha)}$, it means that such K_c cannot be a candidate, and should be discarded.

Rule 2 Given two triples $\big(K_m, K_{\tau_1(\alpha_m)}, \tau_1(\alpha_m)\big)$ and $\big(K_n, K_{\tau_1(\alpha_n)}, \tau_1(\alpha_n)\big)$ which correspond to $Z_{m+8}^{(N)}$ and $Z_{n+8}^{(N)}$ with $0 \leq m \neq n \leq 8$:

- If $\tau_1(\alpha_n) = m$, we can compare $K_{\tau_1(\alpha_n)}$ and K_m:
  - If $K_{\tau_1(\alpha_n)} = K_m$, it indicates that such (K_m, K_n) can be regarded as a candidate;
  - If $K_{\tau_1(\alpha_n)} \neq K_m$, such (K_m, K_n) cannot be a candidate, and should be discarded.
- If $\tau_1(\alpha_n) = \tau_1(\alpha_m)$, we can compare $K_{\tau_1(\alpha_n)}$ and $K_{\tau_1(\alpha_m)}$:
  - If $K_{\tau_1(\alpha_n)} = K_{\tau_1(\alpha_m)}$, it indicates that such (K_m, K_n) can be regarded as a candidate;
  - If $K_{\tau_1(\alpha_n)} \neq K_{\tau_1(\alpha_m)}$, such (K_m, K_n) cannot be a candidate, and should be discarded.

5 The Real-time Inversion Attack on the GMR-2 Cipher

In this section, we present a very efficient and practical attack against the GMR-2 cipher with low time and data complexity. We call this attack the *real-time inversion attack*.

5.1 An Overview of the Inversion Attack

As shown in Fig. 6, we first briefly explain the inversion attack procedure, which is divided into the following three phases:

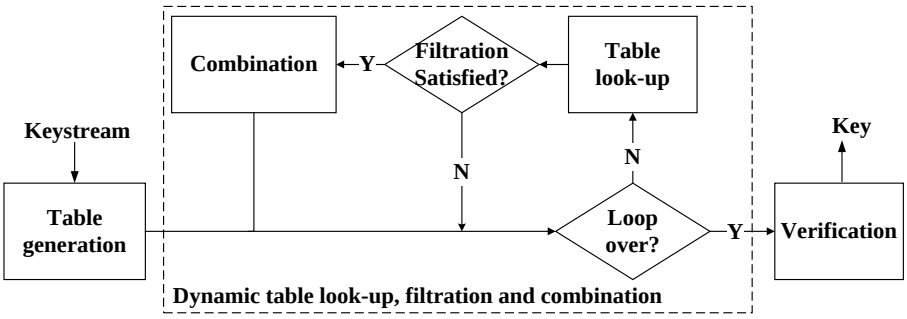


Fig. 6. An overview of the inversion attack procedure

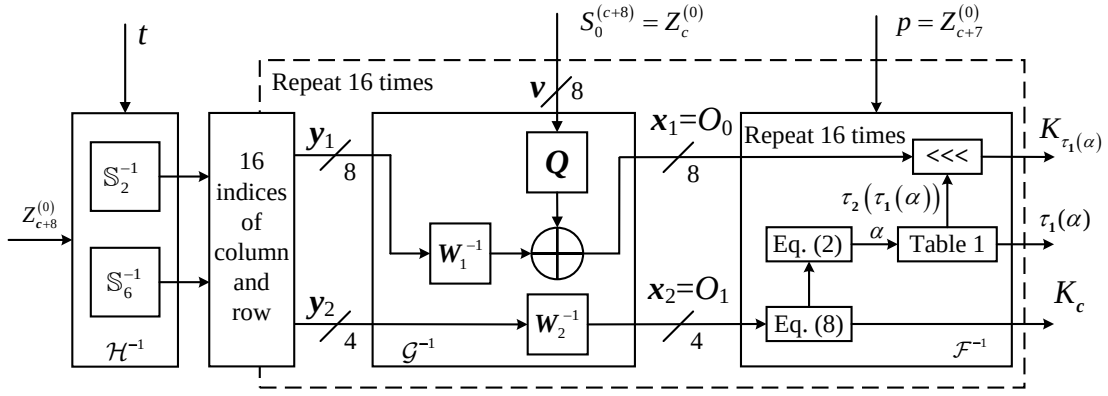


Fig. 7. The Table Generation Procedure

Phase 1: Table generation. Intercept a certain number of keystream bytes (usually only one frame is enoough), then adopt Theorem 2 to generate the 7 lists which map the keystream bytes at the $(c+8)$-th clock with $0 \le c \le 6$ to the original key bytes. Meanwhile, build a virtual list for the 8-th original key byte. We refer these lists as tables.

Phase 2: Dynamic table looks-up, filtration and combination. Look up the tables (8 lists) generated from Phase 1 to obtain candidates for encryption key bytes, adopt Rule $1 \sim 2$ to further filter these candidates, and combine these key bytes that agree with the filter condition and store them in a list. Meanwhile, discard those that do not satisfies the constraints, and backtrack to a proper start-point for new table looks-up. Repeat the steps of table looks-up, filtration and combination until all the candidate keys that meet the constraints of Rule 1 and 2 are found.

Phase 3: Verification. Verify the correctness of those candidate keys obtained in Phase 2 via the intercepted keystream bytes (usually the first 8 bytes of a frame is enough), discard all wrong 8-byte encryption-keys.

5.2 Phase 1: Table Generation

Without loss of generality, assume the frame number of the keystream bytes is $N = 0$, and let $(Z_0^{(0)}, Z_1^{(0)}, \cdots, Z_{14}^{(0)})$ denote the known 15 bytes of keystream. To assure that the values of p and $S_0 = \boldsymbol{v}$ are known, we analyze the cipher at the $(c+8)$-th clock with $0 \le c \le 6$.

According to the mechanism of GMR-2 cipher, each keystream byte $Z_{c+8}^{(0)}$ is related with

$$\left(K_c, K_{\tau_1(\alpha)}, \tau_1(\alpha), p, S_0^{(c+8)}, t\right) = \left(K_c, K_{\tau_1(\alpha)}, \tau_1(\alpha), Z_{c+7}^{(0)}, Z_c^{(0)}, c \bmod 2\right),$$

which means that a mapping between $\left(Z_{c+8}^{(0)}, Z_{c+7}^{(0)}, Z_c^{(0)}\right)$ and $\left(K_c, K_{\tau_1(\alpha)}, \tau_1(\alpha)\right)$ can be established in case c is known. Thus, from the known keystream $\left(Z_0^{(0)}, Z_1^{(0)}, \cdots, Z_{14}^{(0)}\right)$, we can obtain 7 groups of $\left(Z_{c+8}^{(0)}, Z_{c+7}^{(0)}, Z_c^{(0)}\right)$ with $0 \le c \le 6$, and each group can be used to build 256 possible values of triple $\left(K_c, K_{\tau_1(\alpha)}, \tau_1(\alpha)\right)$ based on Theorem 2.

To make a better explanation, one can refer the table generation procedure in Fig. 7. During this phase, for each group of $\left(Z_{c+8}^{(0)}, Z_{c+7}^{(0)}, Z_c^{(0)}\right)$ with $0 \le c \le 6$, the following steps are performed:

1. Look up the two S-boxes to obtain the 16 values of $(\boldsymbol{y}_1, \boldsymbol{y}_2)$ through the keystream byte $Z_{c+8}^{(0)}$ and the toggle-bit t;

2. Calculate the corresponding values of $(\boldsymbol{x}_1, \boldsymbol{x}_2)$ via Eq. 7 for a given $(\boldsymbol{y}_1, \boldsymbol{y}_2)$ from step (1), this also corresponds to O_0 and O_1;
3. Find 16 different values of K_c for a given O_1 according to the Eq. 8, and then get the related values of $K_{\tau_1(\alpha)}$ and $\tau_1(\alpha)$ according to Theorem 1, which yields 16 triples $(K_c, K_{\tau_1(\alpha)}, \tau_1(\alpha))$;
4. Repeat step (2) and step (3) for 16 different values of $(\boldsymbol{y}_1, \boldsymbol{y}_2)$, thereby yield 256 triples $\left(K_c, K_{\tau_1(\alpha)}, \tau_1(\alpha)\right)$ that are stored in a list denoted by $\mathcal{L}_c$ in which K_c is sorted in ascending order.

It should be noted that the above table generation procedure cannot deduce any more information for the 8-th original key byte K_7 from the known keystream $\left(Z_0^{(0)}, Z_1^{(0)}, \cdots, Z_{14}^{(0)}\right)$, *i.e.*, we can only assume that the candidates for K_7 ranges from 0 to 255, but the corresponding values of $K_{\tau_1(\alpha)}$ and $\tau_1(\alpha)$ are not available. Thus we build a virtual list for K_7 ranging from 0 to 255 but with empty values for $K_{\tau_1(\alpha)}$ and $\tau_1(\alpha)$. In total, we generate 8 lists and each list is stored with 256 triples. These 8 lists are denoted by

$$\{\mathcal{L}_0, \mathcal{L}_1, \mathcal{L}_2, \mathcal{L}_3, \mathcal{L}_4, \mathcal{L}_5, \mathcal{L}_6, \mathcal{L}_7\}.$$

5.3 Phase 2: Dynamic Table Looks-up, Filtration and Combination

Now we have generated 8 lists from Phase 1, however, if we simply try exhaustive search using these list without any strategy, there will be no advantage compared with the brute force attack. Thus, before describing our proposed *inversion attack* strategy, we first introduce the following two concepts "key chain" and "valid key chain" based on the 8 lists generated from Phase 1.

Definition 1. ***(Key Chain)*** *A sequence of ordered key bytes*

$$((i_1, K_{i_1}), (i_2, K_{i_2}), \cdots, (i_l, K_{i_l})),$$

where i_j is the index (subscript) for K_{i_j} $(1 \le j \le l)$, is called a key chain with length of l bytes if it satisfies the following condition: for every $1 \le m \le l-1$, there exists a list $\mathcal{L}_{i_m}$ such that $(K_{i_m}, K_{i_{m+1}}, i_{m+1}) \in \mathcal{L}_{i_m}$. For convince, we simply use

$$K_{i_1} \to K_{i_2} \to \cdots \to K_{i_l}$$

to denote this key chain, where K_{i_1} is the starting node and K_{i_l} is the ending node.

Definition 2. ***(Valid Key Chain)*** *A key chain $K_{i_1} \to K_{i_2} \to \cdots \to K_{i_l}$ with length of l bytes is called a valid key chain if it satisfies one of the following conditions:*

1. *There exists an index $i_j \in \{i_1, i_2, \cdots, i_l\}$ such that $(K_{i_l}, K_{i_j}, i_j) \in \mathcal{L}_{i_l}$;*
2. *$i_l = 7$ and there is no other valid key chain that contains the 8-th key byte K_7;*
3. *There already exists a valid key chain with length of* n *bytes: $K_{i'_1} \to K_{i'_2} \to \cdots \to K_{i'_n}$, meanwhile, there exists an index $i_j \in \{i'_1, i'_2, \cdots, i'_n\}$ such that $(K_{i_l}, K_{i_j}, i_j) \in \mathcal{L}_{i_l}$.*

Example Given a key chain with length of three bytes: $K_{i_1} \to K_{i_2} \to K_{i_3}$, the following three cases imply three kinds of valid key chains:

- there exists an index $i_j \in \{i_1, i_2, i_3\}$ such that $(K_{i_3}, K_{i_j}, i_j) \in \mathcal{L}_{i_3}$, as show in Fig. 8(a)
- $i_3 = 7$ and there is no other valid key chain that contains K_7, as show in Fig. 8(b)
- there already exists a valid key chain with length of two bytes: $K_{i'_1} \to K_{i'_2}$, meanwhile, there exists an index $i_j \in \{i'_1, i'_2\}$ such that $(K_{i_3}, K_{i_j}, i_j) \in \mathcal{L}_{i_3}$, as show in Fig. 8(c)

According to the definition, for GMR-2 cipher, the minimum length of a valid key chain is one byte, meaning the key byte is associated with itself. While the maximum length is eight bytes, meaning all the eight key bytes are connected in one chain. Moreover, all valid key chains must be disjoint with each other. Therefore, an 8-byte encryption-key can be divide into at most 8 valid key chains, each containing just one key byte, or at least 1 valid key chain, containing the whole 8 key bytes.

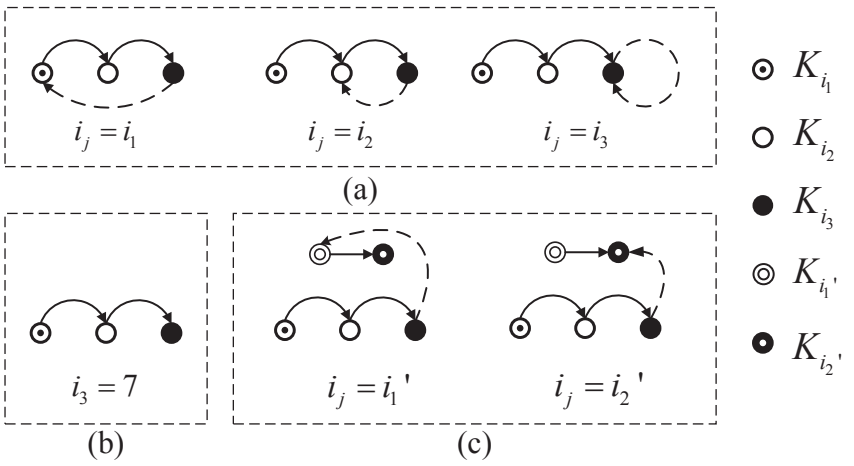


Fig. 8. The Diagram of the Links for Valid Key Chains in the Example

Table 2. Definitions of the variables and candidate sets

Variable	Definition	Initialization
$\mathcal{R}$	The $(i-1)$ valid key chains obtained before: $\mathcal{R} = \{\gamma_1, \gamma_2, \cdots, \gamma_{i-1}\}$.	$\emptyset$
Δ	The key chain currently being looked up: $\Delta = \gamma_i = \left(\delta_{i_1}^{(i)} \to \delta_{i_2}^{(i)} \to \cdots \to \delta_{i_{l-1}}^{(i)}\right)$.	$\emptyset$
$\Gamma = \{\Gamma_1, \Gamma_2\}$	The set of indices(subscripts) for the key bytes that has been obtained by table looks-up, where Γ_1 corresponds to the key bytes in $\mathcal{R}$, and Γ_2 for key bytes in Δ.	$\emptyset$
KC	The candidate set of the complete 8-byte encryption-keys.	$\emptyset$
(c, K_c)	Query point, querying the K_c-th row in the c-th list $\mathcal{L}_c$, it is also used as the control parameter for ending Phase 2.	$(0, 0)$

Main Idea of Phase 2. Using the concept of valid key chain, Phase 2 can be described as "dynamically seeking all valid key chains (that accord with Rule 1 and 2 by table looks-up and the filtration) and combining them to form candidates for the complete 8-byte encryption key". Let's define three candidate sets $\mathcal{R}$, Δ and KC, an index set Γ and a query point (c, K_c) as in Table 2. Using these symbols, and referring Fig. 9, the second phase of the inversion attack can be briefly explained as follows: (For the detail of the attack procedure of Phase 2, one can refer Algorithm 1 and 2.)

1. Choose a starting node (query point) $(0, K_0)$, and for each possible value of K_0 (ranging from $0 \sim 255$), *dynamically lookup the table (8 lists obtained from Phase 1) in a serialized manner* to build up a key chain Δ, and store the indices (subscripts) for the key bytes obtained in Δ into the set Γ_2. Once Δ becomes "valid" through the filtration, treat $\Delta = \gamma_1$ as the first layer of the valid key chain, and store the key bytes of the chain as well as their indices (subscripts) into $\mathcal{R}$, and copy these indices (subscripts) into Γ_1.
2. Choose a new starting node $\left(\min\left(\overline{\Gamma}\right), K_{\min\left(\overline{\Gamma}\right)}\right)$ where $\min\left(\overline{\Gamma}\right)$ is the minimum subscript for the key bytes $(K_0 \sim K_7)$ that have not been obtained before (through table looks-up), and for each possible value of such key byte (also ranging from $0 \sim 255$), continue to *dynamically lookup the table* and *do the filtration* to build up an i-th layer of a valid key chain $\Delta = \gamma_i$ with $1 \le i \le 8$. Similarly update the sets $\mathcal{R} = \{\gamma_1, \gamma_2, \cdots, \gamma_i\}$, Γ_1 and Γ_2.

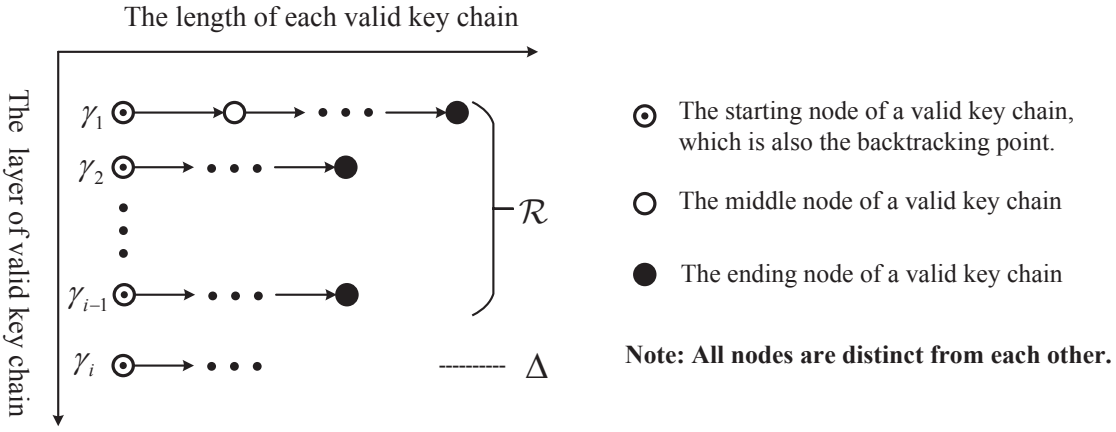


Fig. 9. Dynamic Combination of Valid Key Chains in Phase 2 (The number of valid key chain layers as well as the length of each valid key chain are dynamically changed.)

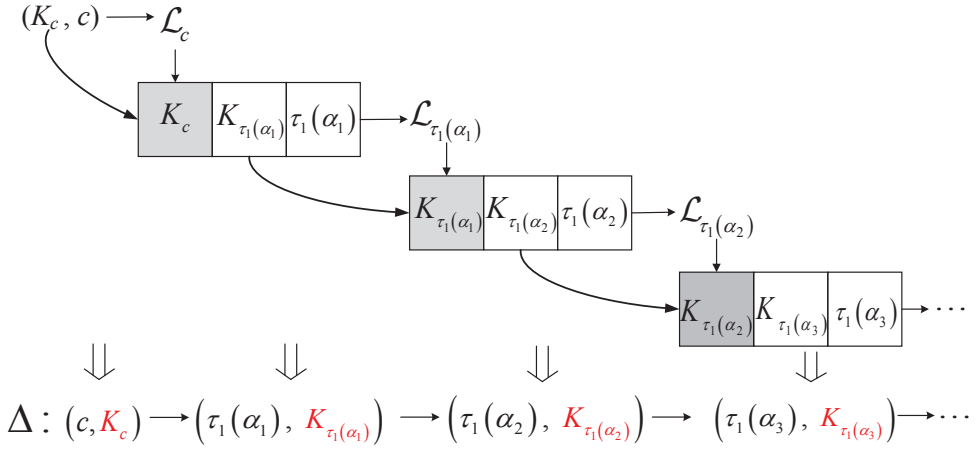


Fig. 10. The Procedure of Dynamic Table Looks-up in Phase 2

3. Check whether all valid key chains in $\mathcal{R}$ exactly cover the whole 8-byte encryption-key, if so, combine these valid key chains, keep them in KC, and backtrack to the starting node of Δ to find a new valid key chain (in order to find new candidate keys), else go back to (2).
4. Repeat Step 1 $\sim$ 3 until all the candidate 8-byte encryption-keys are obtained.

How to dynamically lookup table to build up a key chain? Refer Fig 10, given a query point (c, K_c) as the starting node of a chain Δ, c points to the list $\mathcal{L}_c$, which is then used by the adversary to look up in order to get $\big(K_{\tau_1(\alpha_1)}, \tau_1(\alpha_1)\big)$ that corresponds to its row value K_c. This is followed by a second similar procedure, at this point, we have obtained a middle node $\big(\tau_1(\alpha_1), K_{\tau_1(\alpha_1)}\big)$, then $\tau_1(\alpha_1)$ points to the list $\mathcal{L}_{\tau_1(\alpha_1)}$, which indicates a new result $\big(K_{\tau_1(\alpha_2)}, \tau_1(\alpha_2)\big)$ by looking-up its row value $K_{\tau_1(\alpha_1)}$. Repeat this process, we can further get the next middle nodes $\big(K_{\tau_1(\alpha_3)}, \tau_1(\alpha_3)\big) \cdots$ through the list $\mathcal{L}_{\tau_1(\alpha_2)} \cdots$, thus we will obtain a key chain

$$\Delta = (c, K_c) \to \big(\tau_1(\alpha_1), K_{\tau_1(\alpha_1)}\big) \to \big(\tau_1(\alpha_2), K_{\tau_1(\alpha_1)}\big) \to \big(\tau_1(\alpha_3), K_{\tau_1(\alpha_3)}\big) \to \cdots$$

which is then passed to the filtration procedure to check whether it is valid key chain.

How to do the filtration to get a valid key chain? The purpose of the filtration is to check when the key chain obtained through the table looks-up will be a valid key, this can be done by applying **Rule 1** and **Rule 2** to discard the inconsistence cases. Moreover, during the filtration, we need to do the following extra backtrack steps:

- If the ending node of a key chain disagrees with the constraints of Rule 1 and Rule 2, such chain would not form a valid key chain, then one should backtrack to the starting node of the current key chain $\Delta = \gamma_i$, update $\Gamma_2 \leftarrow \emptyset$, $\Delta \leftarrow \emptyset$, set a new value for this starting node (as the query point), then do a similar procedure of dynamically table looks-up, filtration and combination.
- If the starting node of $\Delta = \gamma_i$ goes beyond the range of $0 \sim 255$, we backtrack to the starting node of the $(i-1)$-th layer of the valid key chain γ_{i-1} in $\mathcal{R}$, and do a similar procedure. Repeat such procedure until we backtrack to the first layer of the valid key chain γ_1. If the starting node of γ_1 is out of the range of $0 \sim 255$, which indicates that all the valid key chains have been found, then we stop the Phase 2 of the inversion attack.

Algorithm 1 Inversion Attack: Phase 2 (Part I)

Input: keystream-related lists $\{\mathcal{L}_0, \mathcal{L}_1, \mathcal{L}_2, \mathcal{L}_3, \mathcal{L}_4, \mathcal{L}_5, \mathcal{L}_6, \mathcal{L}_7\}$.
Output: key candidate set KC.
Initialization: $\mathcal{R} \leftarrow \emptyset$;$\Delta \leftarrow \emptyset$;$\Gamma \leftarrow \emptyset$;$(c, K_c) \leftarrow (0, 0)$; $KC \leftarrow \emptyset$.
repeat
  $\left(\tau_1(\alpha), K_{\tau_1(\alpha)}\right) \leftarrow$ LookUpTable$(c, K_c, \mathcal{L}_c)$;
  if $\tau_1(\alpha) \in \Gamma$ **then** /* Given that the $\tau_1(\alpha)$-th key byte $K_{\tau_1(\alpha)}$ has already existed in the candidate sets, do the filtration using Rule 2 */
    if $\left(\tau_1(\alpha), K_{\tau_1(\alpha)}\right) \in \Delta$ **or** $\left(\tau_1(\alpha), K_{\tau_1(\alpha)}\right)$ *belongs to a certain valid key chain of* $\mathcal{R}$ **then**
      $\Gamma_2 \leftarrow \Gamma_2 \cup \{c\}$; $\Delta \leftarrow \Delta \cup \{(c, K_c)\}$;
      $(c, K_c) \leftarrow$ Combine(Δ); /* The valid key chain obtained at this time agrees with the property (i) or (iii) of Definition 2. */
    else
      $(c, K_c) \leftarrow$ BackTrack(Δ);
    end
  else
    if $\tau_1(\alpha) == c$ **then**
      if $K_{\tau_1}(\alpha) == K_c$ **then** /* Do the filtration using Rule 1. */
        $\Gamma_2 \leftarrow \Gamma_2 \cup \{c\}$; $\Delta \leftarrow \Delta \cup \{(c, K_c)\}$;
        $(c, K_c) \leftarrow$ Combine(Δ); /* The valid key chain obtained at this time agrees with the property (i) of Definition 2. */
      else
        $(c, K_c) \leftarrow$ BackTrack(Δ);
      end
    else
      if $\tau_1(\alpha) == 7$ **then**
        $\Gamma_2 \leftarrow \Gamma_2 \cup \{c, 7\}$; $\Delta \leftarrow \Delta \cup \{(c, K_c), (7, K_7)\}$;
        $(c, K_c) \leftarrow$ Combine(Δ) ; /* The valid key chain obtained at this time agrees with the property (ii) of Definition 2. */
      else /* Continue to find the next node of the current key chain. */
        $\Delta \leftarrow \Delta \cup \{(c, K_c)\}$;
        $(c, K_c) \leftarrow (\tau_1(\alpha), K_{\tau_1}(\alpha))$;
      end
    end
  end
until $K_c > 255$ **and** $c = 0$;
return KC;

Algorithm 2 Inversion Attack: Phase 2 (Part II)

Function `Combine(`Δ`)` `/* Combine a valid key chain or the whole 8-byte key, and return the new starting point (query point) of a new key chain. */`

$\mathcal{R} \leftarrow \mathcal{R} \cup \{\Delta\}$; $\Gamma_1 \leftarrow \Gamma_1 \cup \Gamma_2$;
if Length(Γ_1) == 8 **then** `/* In this case, we have obtained a complete 8-byte candidate key, thus we save it in KC and backtrack to the starting node to build up another candidate key. */`
$KC \leftarrow KC \cup \mathcal{R}$;
$\mathcal{R} \leftarrow \mathcal{R} - \{\Delta\}$; $\Gamma_1 \leftarrow \Gamma_1 - \Gamma_2$;
$(c, K_c) \leftarrow$ BackTrack(Δ);
else `/* If not, we seek the next valid key chain. Here,` $\overline{\Gamma}$ `denotes the set of indices (subscripts) for the key bytes that have not been obtained before. */`
$(c, K_c) \leftarrow \left(\min\left(\overline{\Gamma}\right), 0\right)$;
end
$\Gamma_2 \leftarrow \emptyset$; $\Delta \leftarrow \emptyset$;
return (c, K_c);
end

Function `BackTrack(`Δ`)`

$(c, K_c) \leftarrow$ StartingNodeOf(Δ) ; `/* Backtrack to the starting point of current key chain` γ_i`. */`
$K_c \leftarrow K_c + 1$; `/* Update the key value of the starting node of` γ_i`. */`
if $K_c > 255$ **then**
if $c = 0$ **then**
return (c, K_c);
end
$\Delta \leftarrow \gamma_{i-1}$; $\Gamma_2 \leftarrow$ SubscriptOf(Δ); `/* Update the current key chain. */`
$\mathcal{R} \leftarrow \mathcal{R} - \{\Delta\}$;$\Gamma_1 \leftarrow \Gamma_1 - \Gamma_2$;
$(c, K_c) \leftarrow$ BackTrack(Δ); `/* Backtrack to the starting node of the previous valid key chain` γ_{i-1}`. */`
end
$\Gamma_2 \leftarrow \emptyset$; $\Delta \leftarrow \emptyset$;
return (c, K_c);
end

5.4 Phase 3: Verification

To exclude wrong candidate keys, Phase 3 tests the candidate keys stored in KC one by one, using the first 8 bytes $\left(Z_0^{(0)}, Z_1^{(0)}, \cdots, Z_7^{(0)}\right)$ of the known keystream. For each candidate key, the following steps are performed:

1. Fulfill the key register K with the candidate key, and initialize the shift register S with the known frame number;
2. Clock the cipher 8 times for initialization, and obtain the next 8 bytes keystream;
3. Compare this calculated keystream with the corresponding 8-byte of the intercepted known keystream. If they match, the correct key is found, otherwise, this candidate key is discarded.

6 Experimental Results and Complexity Analysis

In order to verify our proposed attack, in this section, we do some experiments and give the complexity analysis.

6.1 Experimental Results

We carried out 10000 experiments on a 3.3GHz plantform for GMR-2 cipher with random frame numbers and keys. Our results demonstrate that the retrieved encryption-key may not be unique for a known 15-byte keystream at some cases. In other words, there exist multiple encryption-keys corresponding to the

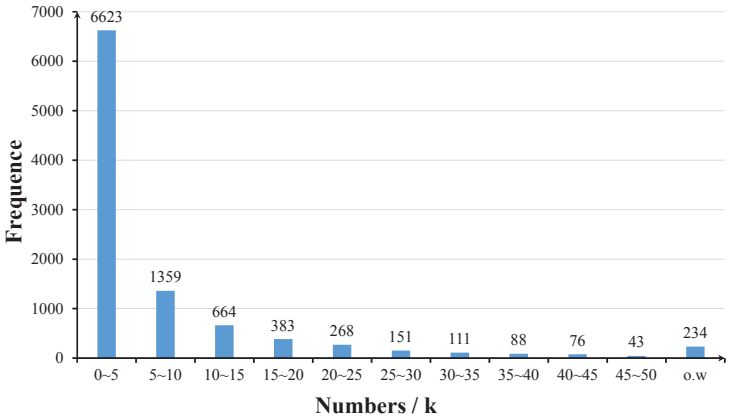


Fig. 11. The frequence distribution of the number of candidate keys (The numbers on horizontal axis are in thousand times, and each interval contains the left value.)

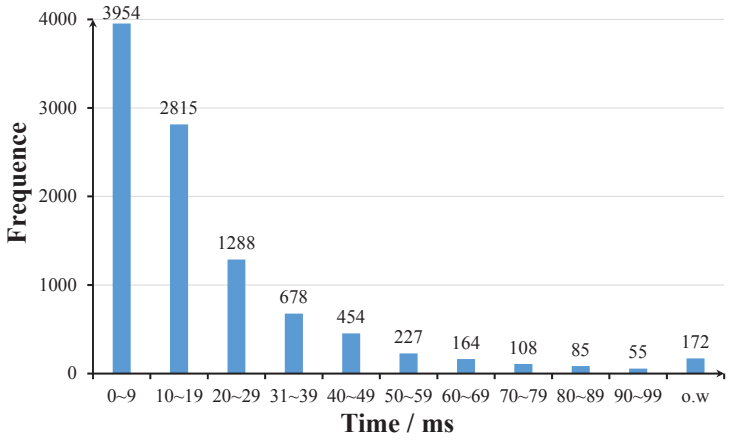


Fig. 12. The frequence distribution of attack time

same 15-byte keystream, and these encryption-keys usually differs one byte from each other. More precisely, each 15-byte keystream indicates 1.03 encryption-keys on average, in which approximately 97.2% of the keystreams indicate a unique encryption-key, and the remaining 2.8% keystreams indicate multiple (at most four) encryption-keys. Thus, to overcome this problem, one additional keystream byte of another frame is needed in these cases, which means that 9 bytes of keystream are totally exploited in the third phase. Therefore, plus one frame of keystream leveraged in Phase 1, the required number of keystream bytes for the whole attack is $15 \sim 16$.

To make a better comparison, the frequence of candidate keys in Phase 2 for each attack are counted with average number 7755 and the distribution is shown in Fig. 11, which shows that one needs to verify 7755 times on average during Phase 3. Meanwhile, the consuming time for each attack are also counted with distribution shown in Fig. 12, which shows that the 8-byte encryption-key can be deduced in around 0.02s on average, where 0.08ms is consumed to generate the table, 3.37ms are consumed to verify the candidates and the rest 16.55ms are consumed by Phase 2.

We also point out that if we perform the forward verification each time an 8-byte candidate key is combined during Phase 2, which means alternating Phase 2 and Phase 3 at the same time, then once an 8-byte candidate key passes the forward verification of the 9 bytes of keystream, the attack can be stopped,

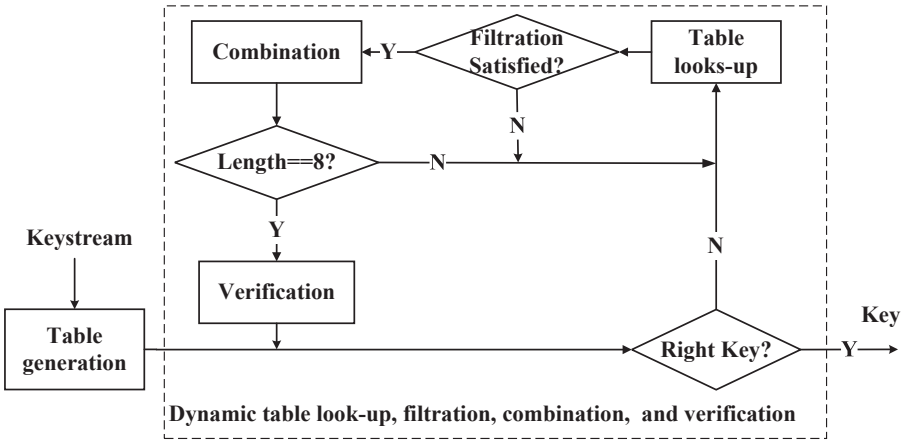


Fig. 13. Optimized inversion attack procedure

in this case, we can accelerate the inversion attack. This *optimized inversion attack* shows that the average number for verifying is reduced to 3980 and the time consumed is about 0.01s on average. The optimized attack procedure is depicted in Fig. 13.

6.2 Complexity Analysis

Time complexity analysis. The time complexity of our inversion attack consists of the time of table generation, dynamic table looks-up and filtration as well as the verification. This can be analysed from the experimental statistics. But for convince, we just focus on the exhaustive search space. As we do verification for $7755 \approx 2^{13}$ times on average, the exhaustive search space is thus about 2^{13}, which could be further reduced to $3980 \approx 2^{12}$ on average when adopting the optimized attack.

Data complexity analysis. The data complexity of our attack is $15 \sim 16$ bytes of keystream. In 10000 experiments, approximately 97.2% of the encryption-keys can be uniquely determined by the 15 bytes of keystream, and the rest (about 2.8%) cases need an extra keystream byte. Thus, $15 \times 97.2\% + 16 \times 2.8\% \approx 15.03$ bytes of keystream are needed to distinguish the right encryption-key from the 2^{13} candidates on average.

Memory complexity analysis. The memory complexity of our attack stems mainly from the table (8 lists) generated in Phase 1. Since each list is filled up with 256 triples $(K_c, K_{\tau_1(\alpha)}, \tau_1(\alpha))$, our attack needs about $256 \times 3 \times 8$ Byte = 6K Bytes of storage space.

7 Conclusions

In this paper, we propose a very efficient, real-time inversion attack against the GMR-2 cipher. It can retrieve the complete 8-byte encryption-key from only 1 frame (15 bytes) of keystream on average, the exhaustive search space can be reduced to about 2^{13} and the memory complexity is 6KB.

Table 3 is the comparison between the known cryptanalytic results and ours, from which we can see that the inversion attack proposed in this paper possesses evident superiority compared with the dynamic guess-and-determine attack and the read-collision based attack. Given one frame (15 bytes) of keystream, one can break the GMR-2 cipher with only 0.02s on a 3.3GHz platform. This again demonstrates that there exists serious security flaws in the GMR-2 cipher, and it is crucial for service providers to upgrade the cryptographic modules[2] of the system in order to provide confidential communication.

[2] Note that the GMR-2 cipher is currently being used in the "IsatPhone Pro" satellite phones.

Table 3. Cryptanalytic results on the GMR-2 cipher

Method	Data	Brute Force Space	Memory	Average Time
Read-Collision Based Technique [4]	$15 \sim 20$ frames	2^{10}	$\sim$	-
Read-Collision Based Technique [4]	$4 \sim 5$ frames	2^{18}	$\sim$	-
Dynamic Guess-and-Determine [13]	1 frame	2^{28}	$\sim$	280s$^{\blacktriangle}$
Inversion Attack (This Paper)	1 frame	2^{13}	6KB	0.02s$^{\triangle}$
Optimized Inversion Attack (This Paper)	1 frame	2^{12}	6KB	0.01s$^{\triangle}$

▲: Experimental plantform: 3.3 GHz platform; Number of experiments: 1000
△: Experimental plantform: 3.3 GHz platform; Number of experiments: 10000

8 Acknowledgment

This work in this paper is supported by the National Nature Science Foundation of China (No: 61402515, 61672530).

References

1. Barkan, P., Biham, E., Keller, N.: Instant Cipher-text Only Cryptanalysis of GSM Encrypted Communication. Journal of Cryptology, 2008, 21,(3), pp. 392-429
2. Biryukov, A., Shamir, A., and Wagner, D.: Real Time Cryptanalysis of A5/1 on a PC. Fast Software Encryption 2000, LNCS 1978, Springer: 1–18.
3. Bogdanov, A., Eisenbarth, T., Rupp, A.: A Hardware Assisted Real-time Attack on A5/2 Without Precomputations. Cryptographic Hardware and Embedded System, 2007, LNCS 4727, Springer: 394-412.
4. Driessen, B., Hund, R., Willems, C., et al.: Don't trust satellite phones: A security analysis of two satphone standards. Security and Privacy (SP), 2012 IEEE Symposium on, Oakland, California, US, 2012: 128-142.
5. Driessen, B., Hund, R., Willems, C., et al.: An experimental security analysis of two satphone standards. ACM Transactions on Information and System Security, 2013, 16, (3): 10.
6. Dunkelman O., Keller N., and Shamir A.: A Practical-Time Attack on the A5/3 Cryptosystem Used in Third Generation GSM Telephony. In CRYPTO 2010, LNCS 6223, Springer: 393–410.
7. ETSI TS: GEO-Mobile Radio Interface Specifications, 2001.
8. Golic, J.D.: On the security of nonlinear filter generators. Fast Software Encryption. Cambridge, UK, February 1996, LNCS 1039, Springer: 173–188.
9. Golic, J.D., Clark, A., and Dawson, E.: Inversion Attack and Branching. Australasian Conference on Information Security and Privacy, 1999, LNCS 1587, Springer: 99–102.
10. Golic, J.D., Clark, A., and Dawson, E.: Generalized Inversion Attack on Nonlinear Filter Generators. IEEE Trans. On Computers, 2000, 49, (10): 1100–1109.
11. Kircanski, A. and Youssef, Amr M.: On the sliding property of SNOW 3G and SNOW 2.0. IET Information Security, 2011, 5(4): 199–206.
12. Li, L., Liu, X., Wang, Z., et al.: An improved attack on clock-controlled shift registers based on hardware implementation. Sci China Inf Sci, 2013, 56: 112107(10).
13. Li, R., Li, H., Li, C., et al.: A Low Data Complexity Attack on the GMR-2 Cipher Used in the Satellite Phones. Fast Software Encryption, 2013, LNCS 8424, Springer: 485–501.
14. Wu, H., Huang, T., and Nguyen, P., et al.: Differential Attacks against Stream Cipher ZUC. ASIACRYPT 2012, LNCS 7658, Springer: 262–277.
15. Zhang, B., Xu, C., and Meier, W.: Fast correlation attacks over extension fields, large-unit linear approximation and cryptanalysis of SNOW 2.0. CRYPTO 2015, Part I, LNCS 9215, Springer: 643–662.
16. Zhou, C., Feng, X., and Lin, D.: The Initialization Stage Analysis of ZUC v1.5. Cryptology and Network Security, 2011, LNCS 7092, Springer: 40–53.